There are principles giving rise to intelligence (machine, human or animal) via learning, simple enough that they can be described compactly, similarly to the laws of physics, i.e., our intelligence is not just the result of a huge bag of tricks and pieces of knowledge, but of general mechanisms to acquire knowledge.
Learning Multiple Levels of Abstraction

- The big payoff of deep learning is to facilitate learning higher levels of abstraction

- Higher-level abstractions can **disentangle the factors of variation**, which allows much easier generalization and transfer

(Bengio & LeCun 2007)
Deep Learning AI Breakthroughs

Computers have made huge strides in

perception,
manipulating language,
games, reasoning, ...

A woman is throwing a frisbee in a park.
A dog is standing on a hardwood floor.

Person
Dog
Chair
2010-2012: breakthrough in speech recognition

Source: Microsoft
2012-2015: breakthrough in computer vision

- Graphics Processing Units (GPUs) + 10x more data
- 1,000 object categories
- Facebook: millions of faces

2015: human-level performance

Use of Deep Learning over Conventional Computer Vision

ImageNet Accuracy Still Improving

Top-5 Classification task

<table>
<thead>
<tr>
<th>Year</th>
<th>U. Toronto</th>
<th>U. Toronto</th>
<th>Google</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>74.2</td>
<td>84.7</td>
<td>88.3</td>
<td>93.3</td>
</tr>
<tr>
<td>2012</td>
<td>84.7</td>
<td>88.3</td>
<td>93.3</td>
<td>96.4</td>
</tr>
<tr>
<td>2013</td>
<td>88.3</td>
<td>93.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>93.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>96.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DEEP LEARNING REVOLUTIONIZING MEDICAL RESEARCH

Detecting Mitosis in Breast Cancer Cells
— IOSIA

Predicting the Toxicity of New Drugs
— Johannes Kepler University

Understanding Gene Mutation to Prevent Disease
— University of Toronto
Medical Image Classification
Clinical Validation: Optical Colonoscopy

World’s first real-time colon polyp malignancy determination from unmodified endoscope raw video with deep learning

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imagia</td>
<td>&gt; 90%, real time</td>
</tr>
<tr>
<td>GI Experts (Key Opinion Leaders)*</td>
<td>~ 90%</td>
</tr>
<tr>
<td>GI Doctors Trained by KOLs*</td>
<td>~ 75%</td>
</tr>
</tbody>
</table>

*(D. Rex, 2015)

Well received by expert clinicians and industry at many conferences, including Digestive Disease Week 2016
GAN: Adversarial generative framework between D and G

Goal: generate from samples from noise, z, transformed by a function G, such that \( p_g(x) \) is close (equal) to a target distribution \( p_d(x) \).

Introduce a discriminator D

(Original) D maximizes the value function (min-max game):

At the optimal discriminator (maximizing \( V \)), minimizing \( V \) amounts to minimizing JSD between \( p_g(x) \) and \( p_d(x) \).

Train purely through back-prop

Produces highly realistic data compared to MLE methods

Does not work naturally with discrete data

Separately Controlling Style & Content

[Input] [Target style] [Output]

[Luan et. al., 2017]
Computers become Creative with Deep Generative Models

- Progress in unsupervised generative neural nets allows them to synthesize a diversity of images, sounds, and text imitating unlabeled images, sounds, or text.

![Diagram of a Generative Adversarial Network (GAN)](image)

- **Generator Network** takes a **Random Vector** and produces a **Fake Image**.
- **Discriminator Network** takes both **Fake Image** and **Real Image** and determines its authenticity.

**Figure 5:** 1024 × 1024 images generated using the CELEB-A-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations. On the right, two images from an earlier megapixel GAN by Marchesi (2017) show limited detail and variation.

**Figure 6:** Visual quality comparison in LSUN BEDROOM; pictures copied from the cited articles.

In order to demonstrate that our contributions are largely orthogonal to the choice of a loss function, we have also trained the same network using LSGAN loss instead of WGAN-GP loss. Figure 1 shows six examples of 1024 × 2 images produced using our method using LSGAN. Further details of this setup are given in Appendix B.

6.4 **LSUN RESULTS**

Figure 6 shows a purely visual comparison between our solution and earlier results in LSUN BEDROOM. Figure 7 gives selected examples from seven very different LSUN categories at 256 × 2. A larger, non-curated set of results from all 30 LSUN categories is available in Appendix G, and the video demonstrates interpolations. We are not aware of earlier results in most of these categories, and while some categories work better than others, we feel that the overall quality is high.

6.5 **CIFAR10 INCEPTION SCORES**

The best inception scores for CIFAR10 (10 categories of 32 × 32 RGB images) we are aware of are 7.90 for unsupervised and 8.87 for label conditioned setups (Grinblat et al., 2017). The large
Intelligence Needs Knowledge

• **Learning:**
  powerful way to transfer knowledge to intelligent agents

• Failure of classical symbolic AI: a lot of knowledge is
  intuitive, difficult to put in rules & facts, not
  consciously accessible

• Solution: get knowledge from data & experience
Machine Learning, AI & No Free Lunch

• Five key ingredients for ML towards AI
  
  1. Lots & lots of data
  2. Very flexible models
  3. Enough computing power
  4. Computationally efficient inference
  
  5. Powerful priors that can defeat the curse of dimensionality
ML 101. What We Are Fighting Against: The Curse of Dimensionality

To generalize locally, need representative examples for all relevant variations!

Classical solution: hope for a smooth enough target function, or make it smooth by handcrafting good features / kernel
Bypassing the curse of dimensionality

We need to build compositionality into our ML models

Just as human languages exploit compositionality to give representations and meanings to complex ideas

Exploiting compositionality can give an exponential gain in representational power

Distributed representations / embeddings: feature learning

Deep architecture: multiple levels of feature learning

Prior assumption: compositionality is useful to describe the world around us efficiently
Distributed Representations: The Power of Compositionality – Part 1

- Distributed (possibly sparse) representations, learned from data, can capture the **meaning** of the data and state
- Parallel composition of features: can be exponentially advantageous

![Image](image_url)
Each feature can be discovered without the need for seeing the exponentially large number of configurations of the other features

- Consider a network whose hidden units discover the following features:
  - Person wears glasses
  - Person is female
  - Person is a child
  - Etc.

If each of $n$ feature requires $O(k)$ parameters, need $O(nk)$ examples

Non-parametric methods would require $O(n^d)$ examples
Hidden Units Discover Semantically Meaningful Concepts

- **Network trained to recognize places, not objects**

<table>
<thead>
<tr>
<th>People</th>
<th>Lighting</th>
<th>Tables</th>
<th>Animals</th>
<th>Seating</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="People" /></td>
<td><img src="Image" alt="Lighting" /></td>
<td><img src="Image" alt="Tables" /></td>
<td><img src="Image" alt="Animals" /></td>
<td><img src="Image" alt="Seating" /></td>
</tr>
</tbody>
</table>

- Fireplace (J=5.3%, AP=22.9%)
- Wardrobe (J=4.2%, AP=12.7%)
- Billiard table (J=3.2%, AP=42.6%)
- Building (J=14.6%, AP=47.2%)
- Bed (J=24.6%, AP=81.1%)
- Mountain (J=11.3%, AP=47.6%)
- Sofa (J=10.8%, AP=36.2%)
- Washing machine (J=3.2%, AP=34.4%)
Deep Learning: Learning an Internal Representation

- Unlike other ML methods with either
  - no intermediate representation (linear)
  - or fixed (generally very high-dimensional) intermediate representations (SVMs, kernel machines)

- What is a good representation? Makes other tasks easier.
Automating Feature Discovery

- Hand-designed program
- Mapping from features
- Most complex features
- Simplest features
- Rule-based systems
- Classic machine learning
- Representation learning
- Deep learning
Learning multiple levels of representation

Successive model layers learn deeper intermediate representations

(Lee, Largman, Pham & Ng, NIPS 2009)
(Lee, Grosse, Ranganath & Ng, ICML 2009)

Parts combine to form objects

Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction
Why Multiple Layers? The World is Compositional

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- **Image recognition**: Pixel $\rightarrow$ edge $\rightarrow$ texton $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object
- **Text**: Character $\rightarrow$ word $\rightarrow$ word group $\rightarrow$ clause $\rightarrow$ sentence $\rightarrow$ story
- **Speech**: Sample $\rightarrow$ spectral band $\rightarrow$ sound $\rightarrow$ ... $\rightarrow$ phone $\rightarrow$ phoneme $\rightarrow$ word
Exponential advantage of depth

- Expressiveness of deep networks with piecewise linear activation functions: exponential advantage for depth
- *(Montufar et al & Bengio, NIPS 2014)*
- Number of pieces distinguished for a network with depth $L$ and $n_i$ units per layer is at least

$$\left( \prod_{i=1}^{L-1} \left[ \frac{n_i}{n_0} \right]^{n_0} \right) \sum_{j=0}^{n_0} \binom{n_L}{j}$$

or, if hidden layers have width $n$ and input has size $n_0$

$$\Omega \left( \frac{n}{n_0} \right)^{(L-1)n_0} n^{n_0}$$
Myth busted:

- Local minima dominate in low-D, but saddle points dominate in high-D
- Most local minima are relatively close to the bottom (global minimum error)

(Dauphin et al NIPS’2014, Choromanska et al AISTATS’2015)
Recap: Machine Learning 101

- Family of functions $f_\theta$
- Tunable parameters $\theta$
- Examples $(x,y)$ sampled from unknown data generating distribution $P(x,y)$
- Loss fn $L$ compares target $y$ and output $f_\theta(x)$, returns a number
- Regularizer $R$ (typically depends on $\theta$ but possibly also on $x$ & $y$)
- Training criterion for supervised learning:
  $$C(\theta) = \text{average}_{(x,y) \sim \text{dataset}} L(f_\theta(x), y) + R(\theta, x, y)$$
- Approximate minimization algorithm to search for good $\theta$
Logistic Regression

- Predict the probability of a **category** \( y \), given input \( x \)
  - \( P(Y=y \mid X=x) \)
- Simple extension of linear regression (binary case):
  - \( P(Y=1 \mid X=x) = \text{sigmoid}(b + w \cdot x) \)
- Train by tuning \((b,w)\) to maximize average log-likelihood
  \[
  \text{Average} \left( \log P(Y=y \mid X=x) \right)
  \]
  over training pairs \((x,y)\), by gradient-based optimization
- This is a very **shallow neural network** (no hidden layer)
Hidden units

(from Hugo Larochelle)

- Neuron pre-activation (or input activation):
  \[ a(x) = b + \sum_i w_i x_i = b + w^\top x \]

- Neuron (output) activation
  \[ h(x) = g(a(x)) = g(b + \sum_i w_i x_i) \]

- \(w\) are the connection weights
- \(b\) is the neuron bias
- \(g(\cdot)\) is called the activation function
A neural network = running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs.

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
A neural network = running several logistic regressions at the same time

... which we can feed into another logistic regression function

and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of predicting the targets for the next layer, etc.
A neural network = running several logistic regressions at the same time

• Before we know it, we have a multilayer neural network....
**Multilayer network as universal approximator**

A series of non-linear transformations of the same type but different parameters

A single but large enough hidden layer yields a **universal approximator**

More layers allow representing more complex functions with less parameters

---

Universal approximator property does not guarantee

1. easy optimization (low training error is found)

2. good generalization
Non-linearity = activation function

- Stacking linear layers: like one (factorized) linear layer
- Universal approximator: stack linear+nonlinear transformations
- Many types of non-linearities are possible: activation function
  - E.g. linear, sigmoid, tanh, rectifier (ReLU), softmax

- Breakthrough in 2011: it is much easier to train a deep multilayer network with rectifiers (ReLU) than with sigmoid or tanh, making it possible to train deep nets in a purely supervised way for the first time (Glorot & Bengio AISTATS 2011)
**Topics:** sigmoid activation function
- Squashes the neuron’s pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

\[ g(a) = \text{sigm}(a) = \frac{1}{1 + \exp(-a)} \]

**Topics:** softmax activation function
- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability \( p(y = c | x) \)
- We use the softmax activation function at the output:
  \[ o(a) = \text{softmax}(a) = \left[ \frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^T \]
  - strictly positive
  - sums to one
- Predicted class is the one with highest estimated probability

**Topics:** hyperbolic tangent (“tanh”) activation function
- Squashes the neuron’s pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

\[ g(a) = \text{tanh}(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1} \]

**Topics:** rectified linear activation function
- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

\[ g(a) = \text{relin}(a) = \max(0, a) \]
Supervised training of an MLP by backpropagation

Requires \((X, Y) = (\text{input, target})\) pairs as training data
Iterative training by SGD

Topics: stochastic gradient descent (SGD)

- Algorithm that performs updates after each example
  - initialize $\theta$ ($\theta \equiv \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$ )
  - for N iterations
    - for each training example $(x^{(t)}, y^{(t)})$
      \[
      \Delta = -\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)}) - \lambda \nabla_\theta \Omega(\theta)
      \]
      \[
      \theta \leftarrow \theta + \alpha \Delta
      \]
    
  training epoch = iteration over all examples

- To apply this algorithm to neural network training, we need
  - the loss function $l(f(x^{(t)}; \theta), y^{(t)})$
  - a procedure to compute the parameter gradients $\nabla_\theta l(f(x^{(t)}; \theta), y^{(t)})$
  - the regularizer $\Omega(\theta)$ (and the gradient $\nabla_\theta \Omega(\theta)$ )
  - initialization method
Motivation for backpropagation: gradient-based optimization

- Knowing how a small change of parameters influences loss $L$ tells us how to change the parameters $\theta$

\[
\frac{\partial L}{\partial \theta}
\]

- The gradient $\frac{\partial L}{\partial \theta}$ measures the ratio of error change due to a small parameter change.

- Indicates the best local descent direction!
Why backprop is powerful

- With $n$ parameters need $O(n)$ computations to obtain $L$
- Also need only $O(n)$ computations to obtain gradient by backprop

- Dumb alternative, by finite differences:

$$\frac{\partial L(\theta_i, \theta_{-i})}{\partial \theta_i} \approx \frac{L(\theta_i + \epsilon, \theta_{-i}) - L(\theta_i, \theta_{-i})}{\epsilon}$$

- But that would cost $O(n^2)$ instead of $O(n)$ by backprop!
Confusion on the word BACKPROP

- **Backprop**: the backward accumulation procedure to compute gradients efficiently wrt a scalar (the loss)

- NOT THE SAME THING AS **gradient descent**, nor the MLP architecture.

- Backprop is **not just used for supervised learning**: also for unsupervised learning and RL, with different losses
Back-Prop & Chain Rule

• Compute gradient of example-wise loss wrt parameters, by considering **intermediate values** such as the outputs of neurons

• **Simply applying the derivative chain rule wisely**

\[ z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \]
Chain Rule

Also works if all these quantities are tensors, using the appropriate tensor products.

\[
\begin{align*}
\Delta z &= \frac{\partial z}{\partial y} \Delta y \\
\Delta y &= \frac{\partial y}{\partial x} \Delta x \\
\Delta z &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x \\
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}
\end{align*}
\]
Multiple Paths Chain Rule

\[
\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}
\]
Multiple Paths Chain Rule - General

\[
\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}
\]
Chain Rule in Flow Graph

Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

\[ \{y_1, y_2, \ldots, y_n\} = \text{successors of } x \]

\[
\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}
\]
Backprop in Multi-Layer Net:

How outputs could change to make error smaller
Backprop in Multi-Layer Net:

How $h_2$ could change to make error smaller
Backprop in Multi-Layer Net:

How $h_1$ could change to make error smaller
Backprop in Multi-Layer Net:

How $W_1$ could change to make error smaller
Back-Prop in General Flow Graph

1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

\[ \{y_1, y_2, \ldots, y_n\} = \text{successors of } x \]

\[ \frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x} \]
Back-Prop in Recurrent & Recursive Nets

- Replicate a parameterized function over different time steps or nodes of a DAG
- Output state at one time-step / node is used as input for another time-step / node
Automatic Differentiation

- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.

Easy and fast prototyping
Log-likelihood as loss function

Topics: loss function for classification

- Neural network estimates \( f(x)_c = p(y = c | x) \)
  - we could maximize the probabilities of \( y^{(t)} \) given \( x^{(t)} \) in the training set

- To frame as minimization, we minimize the negative log-likelihood

\[
l(f(x), y) = - \sum_c 1(y=c) \log f(x)_c = - \log f(x)_y
\]

- we take the log to simplify for numerical stability and math simplicity
- sometimes referred to as cross-entropy
Log-Likelihood for Neural Nets

- Estimating a conditional probability $P(Y|X)$
- Parametrize it by $P(Y|X) = P(Y|\omega = f_\theta(X))$
- Loss = $-\log P(Y|X)$
- E.g. Gaussian $Y$, $\omega = (\mu, \sigma)$
  typically only $\mu$ is the network output, depends on $X$

Equivalent to MSE criterion:

$$\text{Loss} = -\log P(Y|X) = \log \sigma + ||f_\theta(X) - Y||^2 / \sigma^2$$

- E.g. Multinoulli $Y$ for classification,
  $\omega_i = P(Y = i|x) = f_{\theta,i}(X) = \text{softmax}_i(a(X))$
  Loss = $-\log \omega_Y = -\log f_{\theta,Y}(X)$
Multiple Output Variables

- If they are conditionally independent (given $X$), the individual prediction losses add up:

$$- \log P(Y|X) = - \log P(Y_1, \ldots Y_k|X) = - \log \prod_i P(Y_i|X) = - \sum_i \log P(Y_i|X)$$

- Likelihood if some $Y_i$'s are missing: just ignore those losses

- If not conditionally independent, need to capture the conditional joint distribution

$$P(Y_1, \ldots Y_k|X)$$

- Example: output = image, sentence, tree, etc.
- Similar to unsupervised learning problem of capturing joint
- Exact likelihood may similarly be intractable, depending on model